学士論文

擬似3次元表面符号による量子計算の効率化

Neary Optimal Quantum Computing
on Pseudo-Three-Dimentional Surface Code

慶應義塾大学
理工学部物理情報工学科

62115799　平井優我

指導教員　　山本直樹　教授

慶應義塾大学
2025 年 2 月

Contents

1 Introduction

Recent years have witnessed experimental demonstrations of fault-tolerant quantum computation (FTQC) on real devices [1][2]. These experiments highlight 2D surface codes as the most promising quantum error correction codes for FTQC due to their high physical error rate threshold. In the 2D surface code, one can perform lattice surgery [3], which enables logical operations between two distinct surface codes, each encoding a logical qubit. Additionally, to perform universal computation, one must implement magic state distillation [4] for non-Clifford gates such as the T gate or the CCZ gate. Magic state distillation is a highly costly operation; therefore, magic state cultivation [5] has recently emerged as an alternative for implementing the T gate with a cost comparable to that of a CNOT gate.

Representative quantum computing platforms include neutral atoms, superconducting circuits, semiconductors or quantum dots, and photonic quantum computers. In this paper, we study a pipeline architecture [6] for semiconductor quantum computers that utilize one-dimensional shuttling operations. A shuttling operation involves moving a qubit, such as an electron in semiconductors, to enable gate operations between two qubits that are initially far apart before the shuttling process. Neutral atom quantum computers also adopt shuttling operations, enabling the realization of a transversal CNOT gate between two distinct codes, each encoding a logical qubit.

In the pipeline architecture, we can realize a pseudo three-dimensional surface code, where 2D surface codes are stacked on top of each other. We discovered that the pseudo 3D surface code can improve routing operations via lattice surgery for gate operations, as it leverages the 3D structure for routing. In the 3D structure, it is possible to create a route in the third dimension even when no route exists on the 2D plane.

We also study the optimal placement of logical qubits made of surface codes on a processor. By using a mechanical model, such as potential energy, we nearly optimize the placement for 2D processors.

The paper is structured as follows. In Section 2, we present the notations used in the subsequent sections. Section 3 describes the stabilizer formalism, which forms the backbone of quantum error correction theory, and Section 4 explains the surface code in terms of the stabilizer formalism. In Section 5, we discuss lattice surgery operations on the 2D surface code and verify these operations using the stabilizer tableau. Section 6 introduces the pipeline architecture, while Section 7 explains the implementation of surface codes within this architecture and demonstrates how the pipeline architecture enables the realization of a pseudo 3D surface code. In Section 8, we introduce a mechanical model to optimize the placement of logical qubits on the 2D or pseudo 3D surface code. Section 9 presents the results, including improvements in routing for the pseudo 3D surface code compared to the 2D surface code, as well as placement optimization. Finally, Sections 10 and 11 provide the conclusions and future directions for the pipeline architecture and placement optimization on the pseudo 3D surface code.

2 Basics of Quantum Computation

2.1 Qubits [7]

The fundamental unit of quantum information is the qubit. Unlike a classical bit, a qubit can exist in a coherent superposition of its two states, denoted as $|0\rangle$ and $|1\rangle$. These fundamental states are physically realized as charge states in superconducting circuits, the spin of an electron in quantum dots, or atomic spin states. An arbitrary state $|\psi\rangle$ for qubits are expressed as:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{1}$$

where $|0\rangle$ and $|1\rangle$ are two orthogonal basis states in hilbert space $\mathcal{H}$ and $|\alpha|^2 + |\beta|^2 = 1$. In the state vector representation, $|0\rangle$ and $|1\rangle$ are commonly expressed as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2}$$

Unlike classical processing, quantum gates acting on the Hilbert space of qubits must conserve the probability of the qubit states and are therefore unitary. we can define individual qubit gate I, X, Y and Z called Pauli gate. These gates have the following properties:

$$\begin{aligned} I|0\rangle &= |0\rangle, & I|1\rangle &= |1\rangle, \\ X|0\rangle &= |1\rangle, & X|1\rangle &= |0\rangle, \\ Y|0\rangle &= -i|1\rangle, & Y|1\rangle &= i|0\rangle, \\ Z|0\rangle &= |0\rangle, & Z|1\rangle &= -|1\rangle \end{aligned} \tag{3}$$

where i is the imaginary unit. Thus, I is called the identity matrix and acts on a qubit trivially. From Eq. (3), one can derive the matrix representations of the I, X, Y and Z gates as:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{4}$$

If we have multiple qubits, for example, two qubits with states $|0\rangle$ and $|1\rangle$, we represent their combined state using the Kronecker product symbol $\otimes$ as $|0\rangle \otimes |1\rangle = |01\rangle$. In the same way, if we have n states of $|0\rangle$, we represent these states as $|00\cdots0\rangle$. For short, we denote it as $|0\rangle^n$. Additionally, we can define mutiple qubits gate G as:

$$G = \bigotimes_{i=1}^{n} P_i = P_1 \otimes P_2 \otimes \cdots \otimes P_n = P_1 P_2 \cdots P_n \tag{5}$$

where P_j is single qubit gate for jth qubit.

2.2 Gates

In Section 2.1, we introduced only the Pauli gates I, X, Y, and Z. The n-qubit Pauli gates are denoted as a group $\mathcal{P}_n$. There exist additional gates, which can be classified as either Clifford or non-Clifford gates. The definitions of Clifford gates and non-Clifford gates are given as follows.
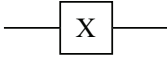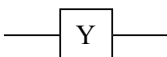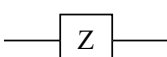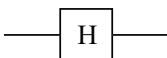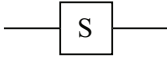
Definition 1. *A group of n-qubit Clifford gates $\mathcal{C}_n$ is defined as:*

$$\mathcal{C}_n = \{V \in U(n) \mid V\mathcal{P}_n V^\dagger \in \mathcal{P}_n\}, \tag{6}$$
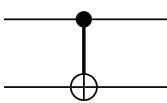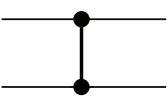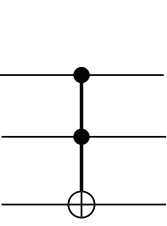
where $U(n)$ denotes the n-qubit unitary group. Non-Clifford gates belong to a group disjoint from the Clifford group.

In the context of quantum computing, we often use various Clifford and non-Clifford gates to construct circuits. The circuit diagrams and matrix representations of these gates are shown in Fig. 1. An operator is synonymous with a gate in the context of quantum computing. One can verify that X, Y, Z, H, S, $CNOT$, and CZ are Clifford gates, while T and CCX are non-Clifford gates, as defined in Definition1.

Table 1

Operator	Gate (in a circuit)	Matrix
Pauli-X (X)	X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli-Y (Y)	Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Pauli-Z (Z)	Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Hadamard (H)	H	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
Phase (S)	S	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
π / 8 (T)	T	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$
Controlled Not ($CNOT$, CX)		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
Controlled Z (CZ)		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$
Toffoli (CCX)		$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$

When we use only Clifford gates, we perform only Clifford operations. However, it has been proven that Clifford operations can be efficiently simulated on a classical computer, as stated in the Gottesman-Knill theorem [8]. To achieve quantum supremacy, we must use non-Clifford operations for universal computation. However, such non-Clifford operations are very costly because they cannot be error-corrected in quantum error correction theory. To implement non-Clifford operations, we must perform magic state distillation [4].

3 Stabilizer Formalism

In this section, we describe the stabilizer formalism[9], which forms the backbone of quantum error correction theory. Before delving into the main discussion, we provide a small case example of error correction and briefly introduce classical error correction theory.

3.1 Introduction to Error Correction

Noise is a great bane of information systems. Whenever we build systems for computation or other purposes, we cannot avoid noise from outside the system. However, even in such environments, classical components perform reliable computation with a failure rate typically below one error in 10^{17} operations. The details of the techniques used to protect against noise in practice are sometimes rather complicated, but the basic principles are easily understood. For example, consider the case when Alice sends Bob a single bit of information through a noisy channel, where the error rate is p. If Alice sends a bit without protection, a bit-flip error, where 0 flips to 1 and vice versa, occurs with a probability of p. As a result, Bob will fail to receive the correct information from Alice with a probability of p. In such cases, we cannot communicate reliably through the noisy channel. Yet, there is a way to address this issue. If Alice introduces redundancy for a bit of information, a process called "encoding," such as:

$$0 \to 000 \quad \text{or} \quad 1 \to 111, \tag{7}$$

Then Alice uses 2 additional bits to send a single bit of information, effectively using two additional channels. In this case, assume Alice sends 000. Each bit can be flipped with a probability of p. Bob then receives:

- No bit flipped, e.g., 000, with a probability of $(1-p)^3$,
- One bit flipped, e.g., 100, 010, 001, with a probability of $3p(1-p)^2$,
- Two bits flipped, e.g., 110, 101, 011, with a probability of $3p^2(1-p)$,
- Three bits flipped, e.g., 111, with a probability of p^3.

Thus, Bob will receive 000, 100, 010, or 001 with a probability of $(1-p)^3 + 3p(1-p)^2$. He then performs a majority vote between 0 and 1, allowing him to restore the information sent by Alice. For reliable communication through encoding, the probability of failure, $3p^2(1-p) + p^3$, must be less than $1/2$. This gives the threshold for reliable communication as $p < 1/2$. Encoding a bit of information into several bits by repeating the original is called a repetition code.

3.2 Error Correction Beyond Classical Ones

In quantum environments, we cannot perform error correction in the same way as described in Section 3.1, because performing a majority vote on an encoded qubit destroys the qubit state, collapsing it into one of the eigenstates of the observable. This is crucial, as in quantum computation, we cannot ignore noise while performing operations. We need to correct errors and perform computational operations simultaneously. If we destroy the states of qubits, we cannot continue the remaining computation. There are also important differences between classical information and quantum information, requiring new ideas to make quantum error-correcting codes possible. In particular, at first glance, we encounter three rather formidable difficulties to address [9]:

- *No cloning*: One might try to implement the repetition code quantum mechanically by duplicating the quantum state three or more times. This is forbidden by the no-cloning theorem.
- *Errors are continuous*: A continuum of different errors may occur on a single qubit. Determining which error occurred in order to correct it would appear to require infinite precision, and therefore infinite resources.
- *Measurement destroys quantum information*: In classical error-correction we observe the output from the channel, and decide what decoding procedure to adopt. Observation in quantum mechanics generally destroys the quantum state under observation, and makes recovery impossible.

Fortunately, these difficulties are not fatal, and we will demonstrate a quantum version of the repetition code to illustrate this. In quantum error correction, there exist two types of errors, in contrast to classical computation: bit-flip errors and phase-flip errors. Bit-flip errors are the same as those in classical computation; for example, they are represented as $X|\psi\rangle$. Phase-flip errors occur when some qubits are acted on by Z, such as $Z|\psi\rangle$. from dualiality of X and Z, we only consider the case that bit-flip errors, and the case of phase-flip errors we can apply the same way as bit-flip. In this demonstration, we will demonstrate the repetition code for bit-flip errors. Suppose we encode the state $|\phi\rangle = a|0\rangle + b|1\rangle$ into a three-qubit space as $|\psi\rangle = a|000\rangle + b|111\rangle$. A useful way to express this is:

$$|0\rangle_L \to |000\rangle, \quad |1\rangle_L \to |111\rangle. \tag{8}$$

Here, $|\cdot\rangle_L$ is called the "logical qubit," while $|\cdot\rangle$ is called the "physical qubit." In this encoding scheme, we now obtain the encoding circuit shown below in Fig. 1.
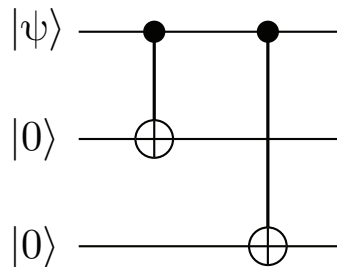

Fig. 1

Using the encoded state, we can obtain syndromes that indicate where an error has occurred. In this

code, these syndromes can be determined by performing Pauli measurements of Z_1Z_2 and Z_2Z_3. For example, when no error has occurred on $|\psi\rangle$, we will obtain syndromes of 1 and 1 from Z_1Z_2 and Z_2Z_3, respectively, due to a straightforward calculation:

$$
\begin{aligned}
Z_1Z_2 \, |\psi\rangle &= aZ_1Z_2 \, |000\rangle + bZ_1Z_2 \, |111\rangle \\
&= a \, |000\rangle + b \, |111\rangle \\
&= |\psi\rangle
\end{aligned}
$$

$$
\begin{aligned}
Z_2Z_3 \, |\psi\rangle &= aZ_2Z_3 \, |000\rangle + bZ_2Z_3 \, |111\rangle \\
&= a \, |000\rangle + b \, |111\rangle \\
&= |\psi\rangle
\end{aligned}
$$

However, if an error has occurred on the first qubit of the three, we will obtain syndromes of -1 and 1 from Z_1Z_2 and Z_2Z_3, respectively, due to a straightforward calculation:

$$
\begin{aligned}
Z_1Z_2 \, |\psi\rangle &= aZ_1Z_2 \, |100\rangle + bZ_1Z_2 \, |011\rangle \\
&= -a \, |100\rangle - b \, |011\rangle \\
&= - \, |\psi\rangle
\end{aligned}
$$

$$
\begin{aligned}
Z_2Z_3 \, |\psi\rangle &= aZ_2Z_3 \, |100\rangle + bZ_2Z_3 \, |011\rangle \\
&= a \, |100\rangle + b \, |011\rangle \\
&= |\psi\rangle .
\end{aligned}
$$

Thus, we can determine that a bit-flip error has occurred on the first qubit and correct the error. In this case, we can correct a single error on any qubit. However, we cannot correct two or three errors. For instance, in the case of two errors on the second and third qubits of the three, we will obtain the following syndromes:

$$
\begin{aligned}
Z_1Z_2 \, |\psi\rangle &= aZ_1Z_2 \, |011\rangle + bZ_1Z_2 \, |100\rangle \\
&= -a \, |011\rangle - b \, |100\rangle \\
&= - \, |\psi\rangle
\end{aligned}
$$

$$
\begin{aligned}
Z_2Z_3 \, |\psi\rangle &= aZ_2Z_3 \, |011\rangle + bZ_2Z_3 \, |100\rangle \\
&= a \, |110\rangle + b \, |100\rangle \\
&= |\psi\rangle .
\end{aligned}
$$

These syndromes are the same as in the former case, so we cannot determine whether a single error has occurred on the first qubit or two errors have occurred on the second and third qubits. in the case of three errors, we will obtain the following syndromes:

$$
\begin{aligned}
Z_1Z_2 \, |\psi\rangle &= aZ_1Z_2 \, |111\rangle + bZ_1Z_2 \, |000\rangle \\
&= a \, |111\rangle + b \, |000\rangle \\
&= |\psi\rangle
\end{aligned}
$$

$$
\begin{aligned}
Z_2Z_3 \, |\psi\rangle &= aZ_2Z_3 \, |111\rangle + bZ_2Z_3 \, |000\rangle \\
&= a \, |111\rangle + b \, |000\rangle \\
&= |\psi\rangle ,
\end{aligned}
$$

which makes it appear as though no errors have occurred, even though errors have actually occurred. In summary, this code can detect and correct one error, can detect two errors but cannot correct them, and cannot detect or correct three errors.

One may find that if a continuous error, such as $|0\rangle \to |0\rangle + |1\rangle$, occurs, we cannot obtain syndromes. Actually, when we obtain syndromes, we perform syndrome measurements using $Z_1 Z_2$ and $Z_2 Z_3$. As a result, the superposition state will be projected to either $|0\rangle$ or $|1\rangle$ with a probability proportional to the coefficients of the superposition state.

In the following section, we refer to the minimum number of errors that cannot be detected as the code distance, often expressed as d, encoded qubits as logical qubits, and the number of them as k, and qubits used to encode logical qubits as physical qubits, with their number often expressed as n. The properties of the code are often denoted as $[[n, k, d]]$. In the case of the former repetition code, its properties are $[[3, 1, 1]]$, but this code cannot correct phase-flip errors. Therefore, we show how to correct both types of errors: bit-flip and phase-flip.

3.3 Stabilizer

Stabilizer formalism is the most fundamental concept in quantum error correction. The stabilizer group is denoted as $\mathcal{S}$, and its elements are Pauli matrices, so $\mathcal{S} \subseteq \mathcal{P}$, but $-I \notin \mathcal{S}$. We also define the centralizer $\mathcal{C}(\mathcal{S})$, whose elements are Pauli matrices that commute with all elements of $\mathcal{S}$. Unlike classical codes, quantum codes are designed for computation while correcting errors, so there exist logical operators that change the states of logical qubits but leave stabilizer states unchanged. These logical operators L exist in $\mathcal{C}(\mathcal{S})\backslash\mathcal{S}$. In the context of Section 3.2, $\mathcal{S} = \{Z_1 Z_2, Z_2 Z_3\}$ and $\mathcal{C}(\mathcal{S}) = \{Z_1, Z_2, Z_3, X_1 X_2 X_3\}$, so in this case, the logical operators are $L_X = X_1 X_2 X_3$ and $L_Z = Z_1$, Z_2, or Z_3. As a side note, Z_1, Z_2, and Z_3 are equivalent logical operators in the quotient group of the stabilizer group.

In quantum error correction theory, there are representative codes, and we will provide an example of one, called the Steane Code [10]. The Steane Code consists of 7 qubits and 6 stabilizer generators, making it an $[[7, 1, 1]]$ code. Stabilizer generators form the basis of the stabilizer group, meaning all elements of the stabilizer group can be expressed as products of stabilizer generators. stabilizer generators and logical operators are shown in Tab. 2.

Table 2

Stabilizer Generators	Logical Operators
$I\ I\ I\ X\ X\ X\ X$	
$I\ X\ X\ I\ I\ X\ X$	
$X\ I\ X\ I\ X\ I\ X$	$L_X = X\ X\ X\ X\ X\ X\ X$
$I\ I\ I\ Z\ Z\ Z\ Z$	$L_Z = Z\ Z\ Z\ Z\ Z\ Z\ Z$
$I\ Z\ Z\ I\ I\ Z\ Z$	
$Z\ I\ Z\ I\ Z\ I\ Z$	

One may wonder how to encode a logical qubit in such codes. Encoding involves preparing a stabilizer state for the encoded qubits, so all you need to do is perform projective measurements of the stabilizer

generators. First, you initialize all physical qubits to $|0\rangle$. This state is already a stabilizer state of the Z stabilizers listed in Tab. 2. Second, you only need to perform projective measurements of the X stabilizers. Consequently, the encoded state $|0\rangle_L$ can be expressed as follows:

$$|0\rangle_L = \prod_i (I + s_i) |0000000\rangle$$

where s_i is a Z stabilizer generator, and the state is post-selected to the $+1$ eigenstates, ignoring the normalization factor. In the same way, $|1\rangle_L$ can be encoded. Thus, if you want to encode an arbitrary state $|\psi\rangle = a |0\rangle + b |1\rangle$, first prepare the Bell state $a |0000000\rangle + b |1111111\rangle$ and perform projective measurements of the Z stabilizer generators. The final states is:

$$|\psi_L\rangle = a \prod_i (I + s_i) |0000000\rangle + b \prod_i (I + s_i) |1111111\rangle .$$

From straightforward calculations, you can confirm that $L_X |0\rangle_L = |1\rangle_L$, $L_Z |1\rangle_L = - |1\rangle_L$, and $L_X L_Z = -L_Z L_X$. By definition, a logical operator cannot be expressed as the product of stabilizer generators.

To perform syndrome measurements, we need to implement multi-body Pauli measurements, which are challenging to directly implement in experiments. Therefore, we require an ancilla qubit for each stabilizer. As shown in Fig. 2, (a) represents a 4-weight Z stabilizer measurement circuit, and (b) represents a 4-weight X stabilizer measurement circuit. The top four qubits are data qubits, and the bottom qubit is the ancilla qubit. In the following sections, we assume these circuits are used for syndrome measurements.
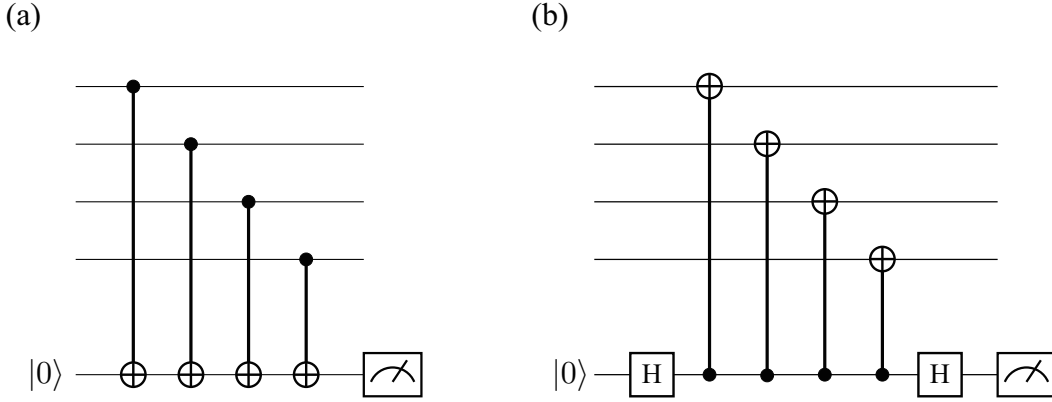
(a)   (b)

Fig. 2

3.4　Error Detction

In this subsection, we will describe how to detect errors using syndrome measurements. Let $|\psi\rangle^t$ be the state of the physical qubits at round t, where "round" indicates the sequence number of the error correction operation. After the round t error correction, some logical operations are performed, followed by round $t + 1$ error correction. Finally, this results in the round $t + 1$ state $|\psi\rangle^{t+1}$. Assume there are no errors in $|\psi\rangle^t$, and that errors occur due to the logical operations between round t and round $t + 1$. In this case, we obtain a syndrome m_j^t of $s_j \in \mathcal{S}$ at round t as:

$$s_j |\psi\rangle^t = m_j^t |\psi\rangle^t .$$

10

And at round $t + 1$, we obtain a syndrome m_j^{t+1} as:

$$s_j \ket{\psi}^{t+1} = m_j^{t+1} \ket{\psi}^{t+1} .$$

In the case where the errors can be detected by s_j, m_j^{t+1} has the opposite sign of m_j^t, so $m_j^{t+1} m_j^t = -1$. On the other hand, if the errors cannot be detected by s_j, m_j^{t+1} has the same sign as m_j^t, so $m_j^{t+1} m_j^t = 1$. From this discussion, we only need to know the product of the syndromes from round t and round $t + 1$.

4   Surface Code

The Surface Code, first introduced by Kitaev [11], is the most promising error correction code for quantum computing. Using this code, universal computation can be performed with magic state distillation. In this section, we will first introduce the stabilizers of the Surface Code on the torus and then on the planar surface.

4.1   Surface Code on the Torus
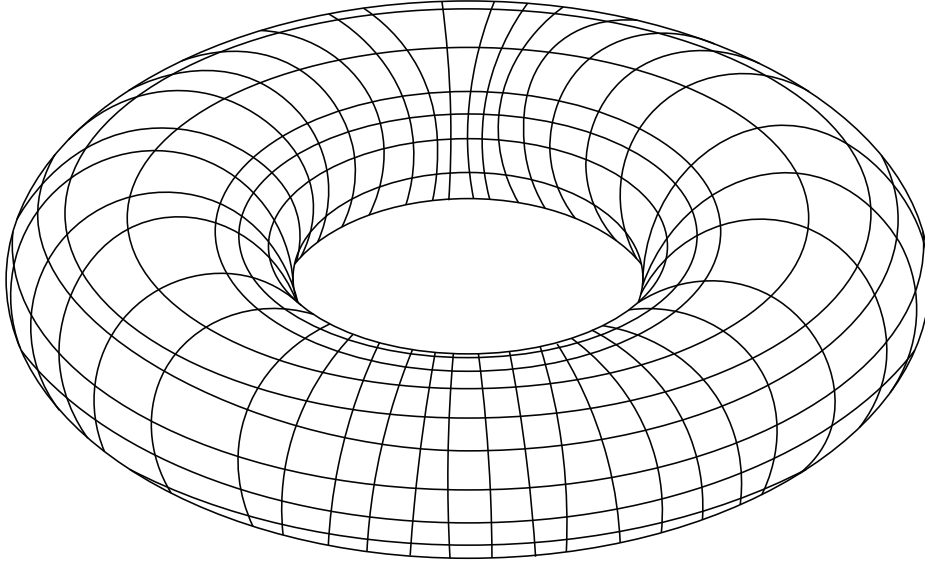
Fig. 3

Usually, the Surface Code is defined on a 1-genus torus, but it can also be defined on an n-genus torus in the same way. The lattice on the torus, shown in Fig. 3, has the following property:

$$V - E + F = 2 - 2g \tag{9}$$

where V is the number of vertices, E is the number of edges, F is the number of faces of the lattice on the torus, and g is the genus. For $g = 1$, the 1-genus case, Eq. 9 equals 0. We now introduce data qubits on the edges, X stabilizers on the vertices, and Z stabilizers on the faces, as shown in Fig. 4. Incidentally, in Fig. 4, the left and right sides are identical, and similarly, the upper and bottom sides are identical. Ancilla qubits for syndrome measurements are allocated at the center of the faces for the Zstabilizers and on the vertices for the X stabilizers.
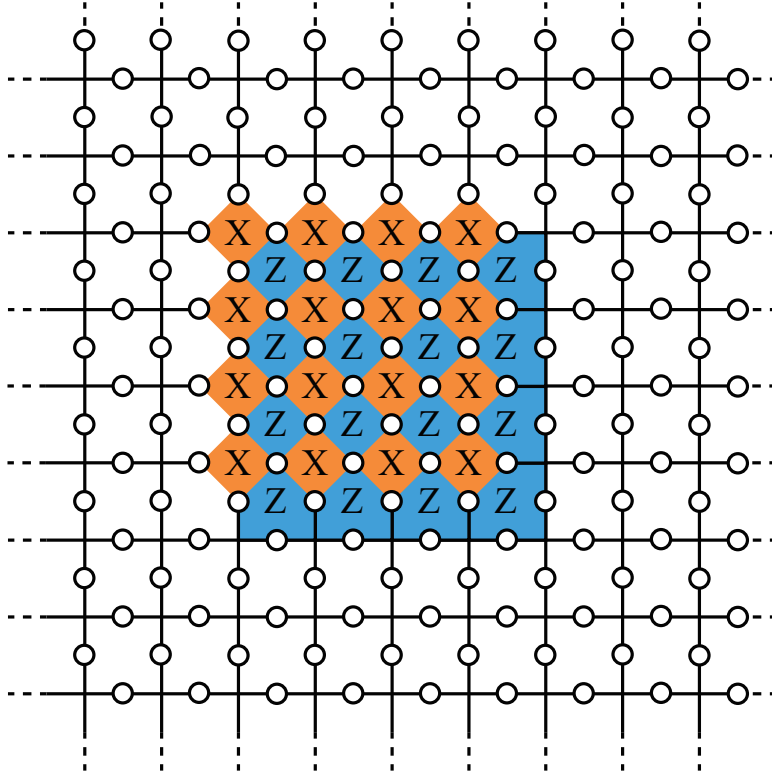
Fig. 4

In Fig. 4, we show some of the stabilizers of the Surface Code, but others exist in the remaining parts of the lattice. Thus, we have $V - 1$ X stabilizer generators because the product of all X stabilizers on the vertices is the identity. From the same discussion, we have $F - 1$ Z stabilizer generators. Therefore, the number of logical qubits k that can be encoded in the Surface Code is:

$$k = E - (V - 1) - (F - 1) = 2 \tag{10}$$

using Eq. 9. From these results, we can identify four logical operators corresponding to the non-trivial cycles on the torus. The logical operators are shown in Fig. 5, where the subscripts 1 and 2 indicate the qubit numbers of the two logical qubits. From Fig. 5, one can confirm that $L_X^i L_Z^i = -L_Z^i L_X^i$, that all logical operators commute with the stabilizers, and that the code distance is $\sqrt{n}$, which is the least weight of a logical operator.

4.2 Surface Code on the Planar

The Surface Code on the planar is different from that on the torus because it has boundaries, as shown in Fig. 6. One can observe that there exist 3-weight stabilizers at the boundaries.
Additionally, the Surface Code on the planar can encode one logical qubit (Fig. 7). In the same way of the torus, one can confirm that code distance of the surface code in Fig. 7 is $\sqrt{n}$ where there exists n qubits. The two boundaries that are connected to each other by the logical Z operator are called rough boundaries, while the remaining boundaries are called smooth boundaries.

There exist excess qubits in the previous Surface Code. As shown in Fig. 8, we can cut along the green line, which contains both data qubits and ancilla qubits, trim off the excess parts, and rotate the lattice by 45 degrees to obtain the rotated Surface Code. With this modification, the code distance
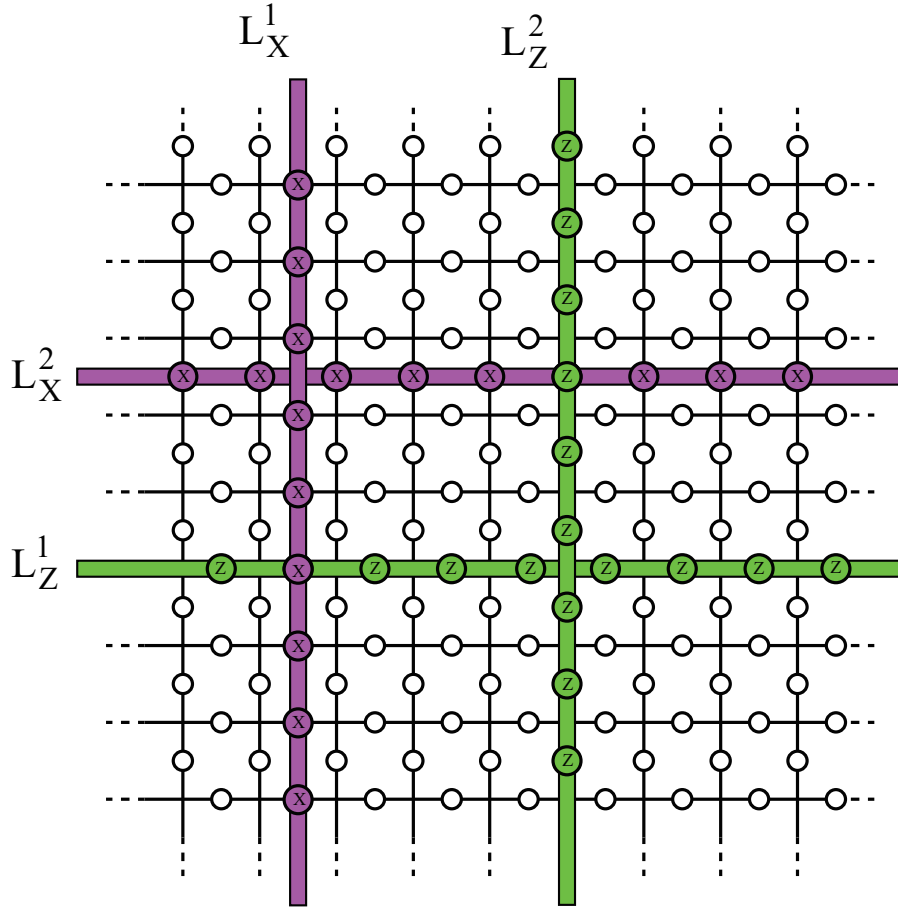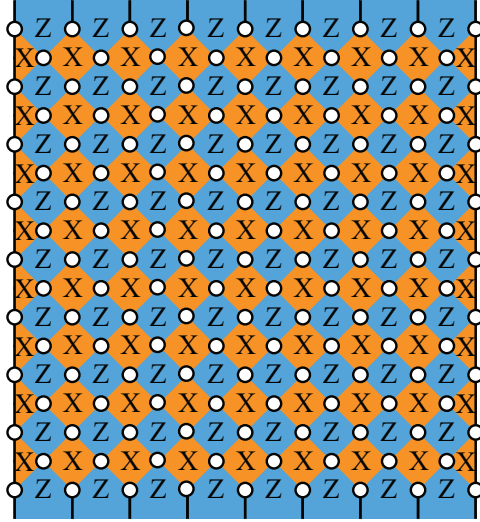
Fig. 5

Fig. 6

of the Surface Code is preserved by defining new logical operators on the rotated Surface Code that connect the rough boundaries or the smooth boundaries.
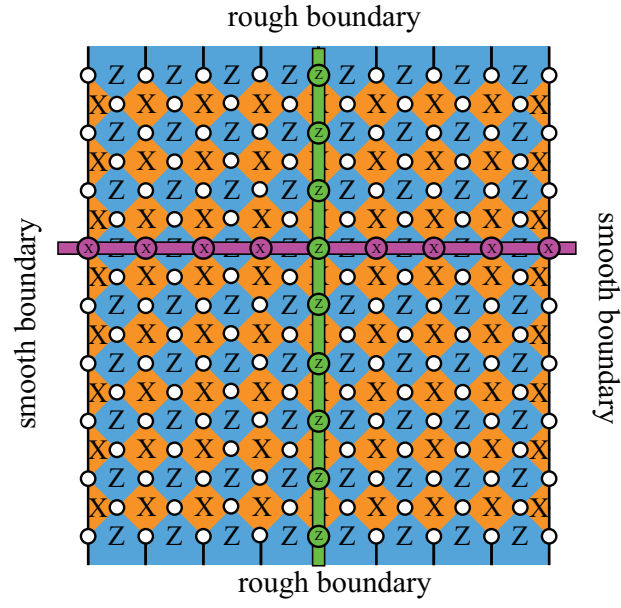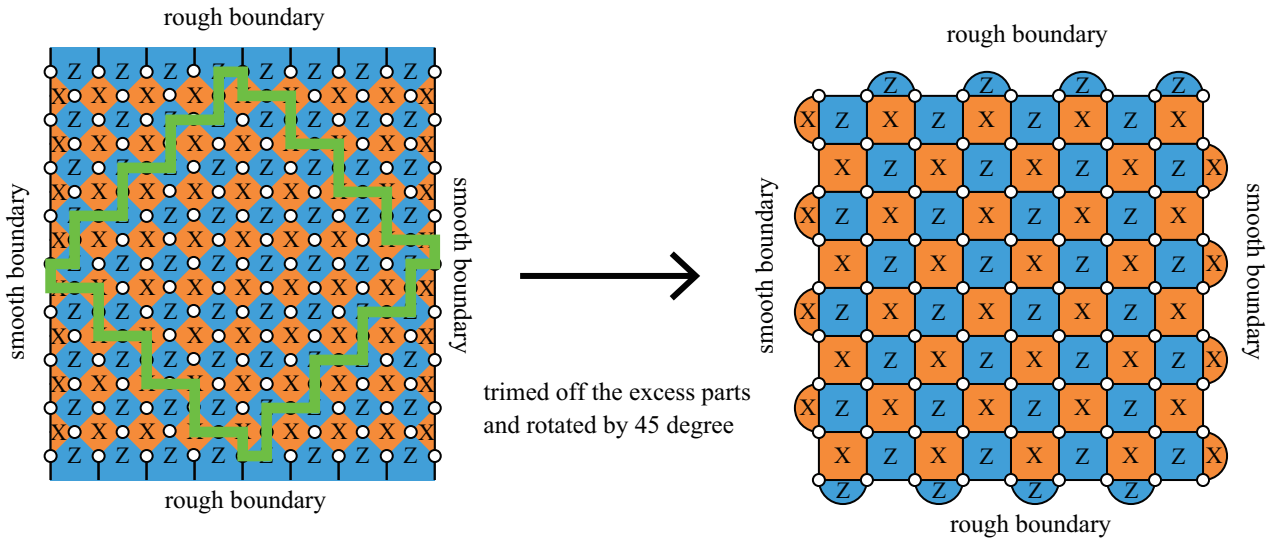
Fig. 7

Fig. 8

5 Lattice Surgery

Lattice Surgery [3] is an operation of code deformation, where a code is transformed into another code and then returned to the initial code, resulting in a change in the logical qubit states. In this section, we will first introduce the lattice surgery operation and then describe a CNOT operation implemented using lattice surgery.

5.1 Merging

The Lattice Surgery operation consists of two operations: Merging and Splitting. In this section, we will first describe the merging operation. There are two types of merging: smooth merging and rough merging. Smooth merging involves merging the smooth boundaries of two Surface Codes, while rough merging involves merging their rough boundaries. Briefly, the procedure for the smooth merging operation is shown in Fig. 9; the rough merging operation can be performed in almost the same way.
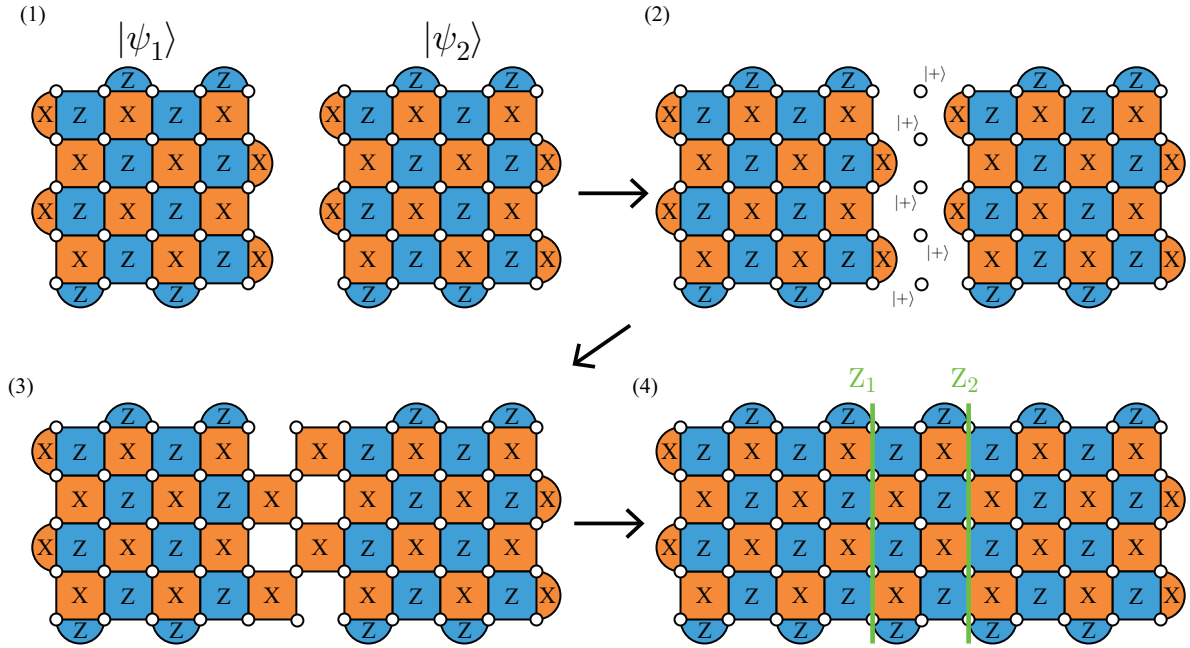

Fig. 9

In the following description of the smooth merging operation, the notations (1), (2), (3), and (4) correspond to (1), (2), (3), and (4) in Fig. 9. In step (1), two arbitrary logical states, $|\psi_1\rangle$ and $|\psi_2\rangle$, encoded by the surface codes, are placed adjacent to each other. In step (2), new data qubits are introduced and initialized in the $|+\rangle$ state between the two logical qubits. By this initialization, 4-weight X stabilizers that connect the two logical states are already established in step (3), so no additional operations are required in step (3). Then, in step (4), we perform the syndrome measurements of Z stabilizers that connect the two logical qubits. The product of all Z stabilizers added in step (4) equals $Z_1 Z_2$, where Z_i is the logical operator of the state $|\psi_i\rangle$. Thus, we can obtain a measurement result $m_{Z_1 Z_2}$ for $Z_1 Z_2$. This operation can be written as:

$$O_{\text{merging}} |\psi_1\rangle |\psi_2\rangle = (I + (-1)^{m_{Z_1 Z_2}} Z_1 Z_2) |\psi_1\rangle |\psi_2\rangle \tag{11}$$

where O_{merging} indicates the merging operation in the equation. We have merged the smooth boundaries of two Surface Codes, but the rough boundaries can be merged in the same way.

5.2 Splitting

In the previous section, we introduced the merging operation of lattice surgery. In this section, we will introduce the splitting operation, which is the opposite of the merging operation.
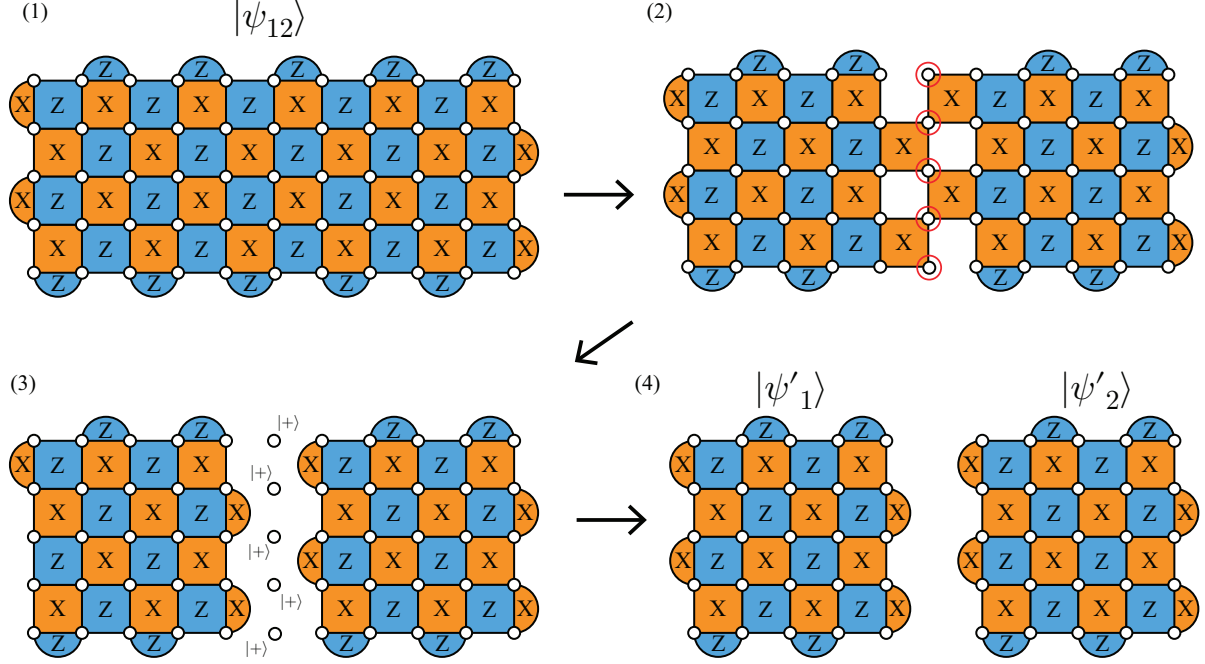

Fig. 10

In the following description of the splitting operation, the notations (1), (2), (3), and (4) correspond to (1), (2), (3), and (4) in Fig. 10. In step (1), we start with a state $|\psi_{12}\rangle$. In step (2), the middle column of qubits is measured in the Pauli-X basis. As a result, some Z stabilizers in the middle of the qubits are lost in step (3). By eliminating the data qubits measured in the Pauli-X basis, we obtain two logical states, $|\psi'_1\rangle$ and $|\psi'_2\rangle$, in step (4). While the measurement results will be used for error correction after the splitting operation, we avoid delving into that aspect here.

5.3 Logical CNOT Gate

In this section, we introduce a logical CNOT gate using the lattice surgery operation described in the previous sections. In quantum error correction theory, a logical CNOT gate is often implemented using measurement-based quantum computation. Within the lattice surgery framework, we can leverage this approach. The CNOT gate implemented by local measurements is shown in Fig. 11.
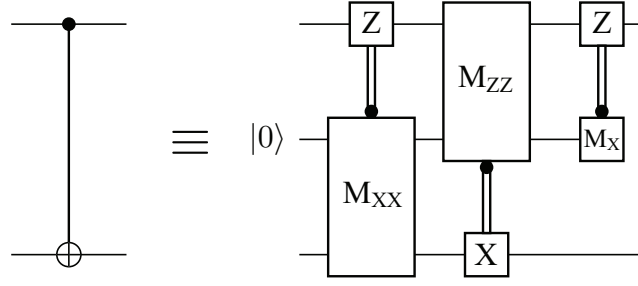
Fig. 11

In the previous sections, we have seen the measurement operation $Z_1 Z_2$ $(X_1 X_2)$ for logical qubits. Ignoring the Pauli correction based on the measurement results, as long as a 2-weight Pauli measurement on the logical qubits can be performed, we can achieve a CNOT gate. The protocol to implement the CNOT gate, where $|\psi_1\rangle$ is the target and $|\psi_2\rangle$ is the control, is shown in Fig. 12. In the following description of the logical CNOT operation, the notations (1), (2), (3), (4), (5), and (6) correspond to (1), (2), (3), (4), (5), and (6) in Fig. 12. In step (1), there are three logical states, $|\psi_1\rangle$, $|\psi_2\rangle$, and $|0\rangle$, referred to as qubit-1, qubit-2, and the ancilla qubit, respectively. All of these states are encoded in the Surface Code. Additionally, unused data qubits for the merging operation exist in the middle of the logical qubits. Then, a rough merging operation is performed between qubit-1 and the ancilla qubit in step (2). In step (3), qubit-1 and the ancilla qubit are roughly split. Next, a smooth merging operation is performed between qubit-2 and the ancilla qubit, followed by smooth splitting between qubit-2 and the ancilla qubit. Finally, a measurement is performed on the ancilla qubit in the Pauli-X basis.
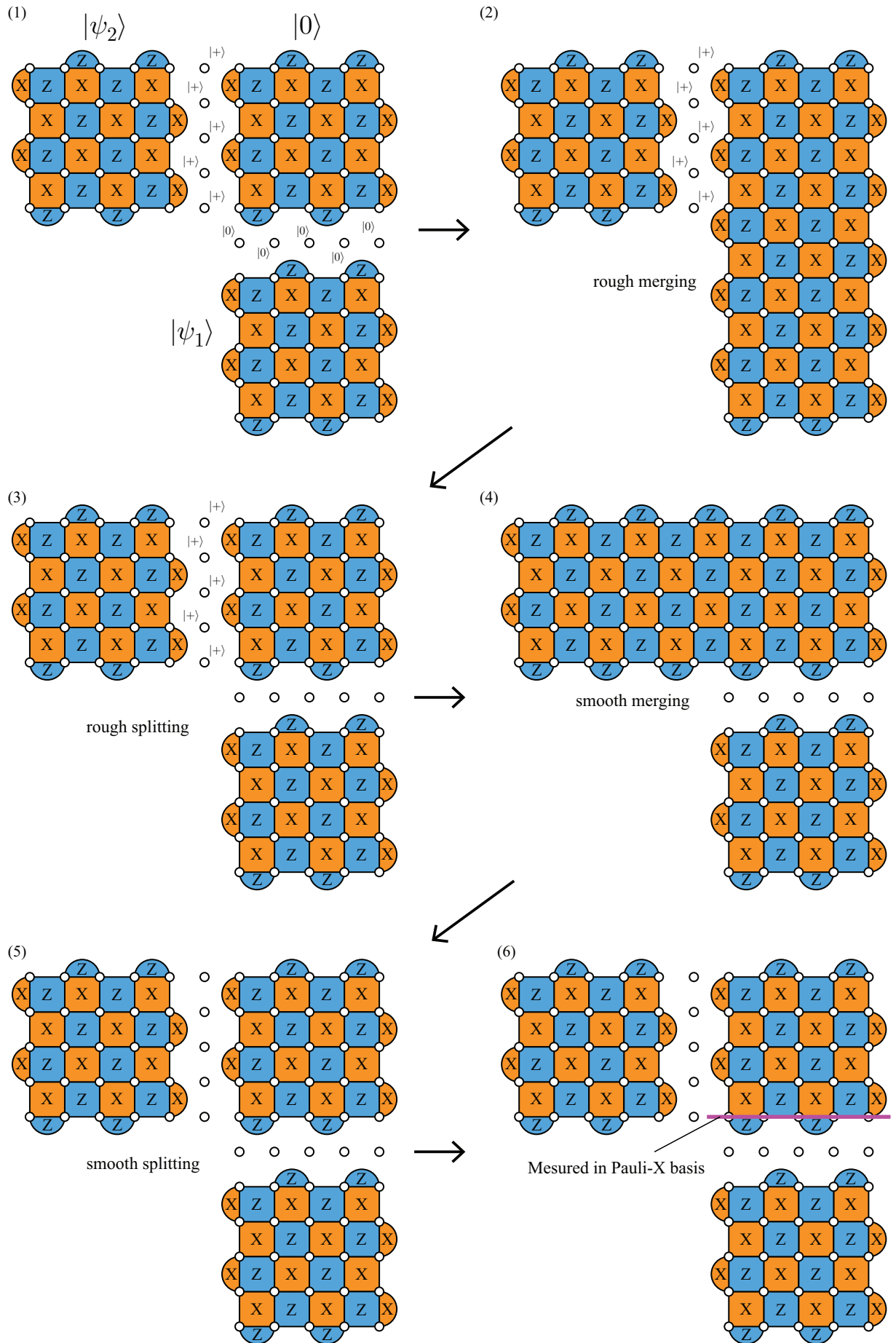
(1) $|\psi_2\rangle$ $|0\rangle$

(2) rough merging

(3) rough splitting

(4) smooth merging

(5) smooth splitting

(6) Mesured in Pauli-X basis

$|\psi_1\rangle$

Fig. 12

18

6 Looped Pipeline Architecture

Many quantum computing platforms are based on a two-dimensional physical layout. We focus on "looped pipelines," [6] which offer the advantages of a three-dimensional lattice while being restricted to two-dimensional space. This architecture leverages qubit shuttling, where qubits are moved around on the chip, enabling long-range interactions between qubits that are far apart.

6.1 Classical Linear Pipeline
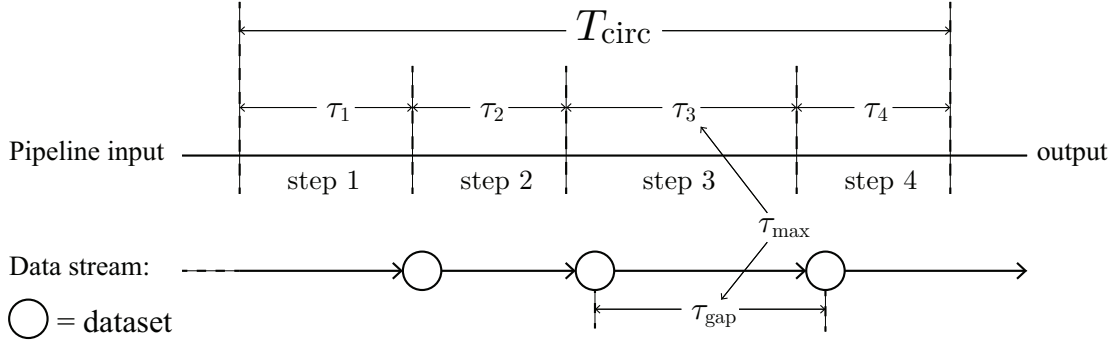
Fig. 13

Let us fist introduce "data-processing pipeline," we divide data-processing circuit into multple steps with each step able to operate only one dataset at a time shown in Fig. 13. Now intead of inputting second dataset after the processing the first dataset on whole circuit, we input it as soon as finishing the process of the first dataset on step 1, similarly all subseaquent datasets and subseaquent steps. In this way, all data-processing steps can be working on different datasets in parallel, increasing the throughput of the system. this is a concept of pipelining. The processing time of the mth step is denoted as τ_m, and suppose there exist M steps in the circuit. Then, the total time T_{circ} taken for the entire circuit to process one dataset is given by:

$$T_{\text{circ}} = \sum_{m=1}^{M} \tau_m. \tag{12}$$

This is the time required for every dataset without pipelining, and it is also the time required to process the first dataset in pipelining. For the second dataset in pipelining, the additional time required for processing is given by:
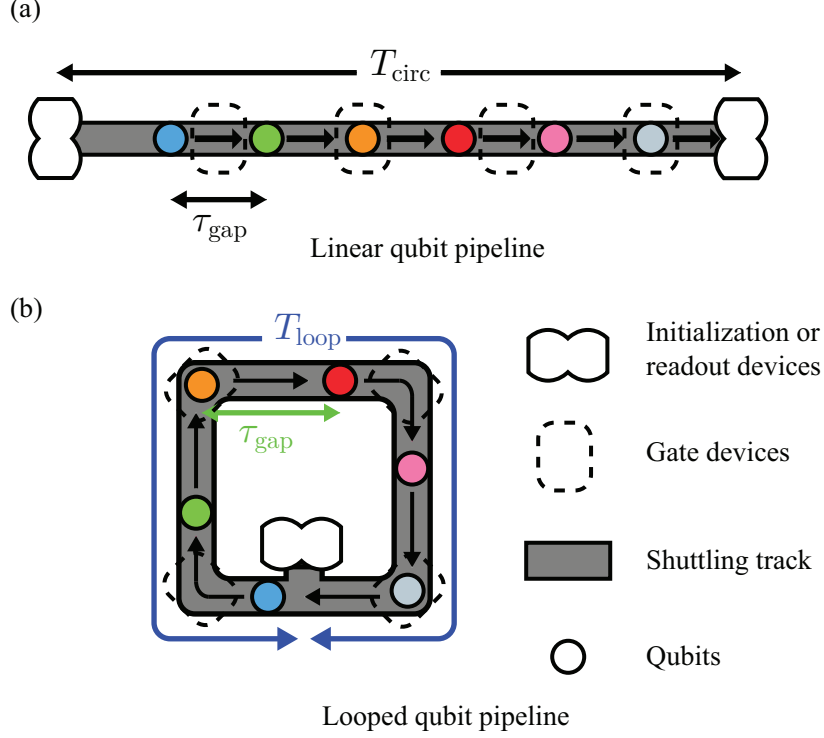
$$\tau_{\text{max}} = \max_i \tau_i. \tag{13}$$

which is called "rate-limiting step." From the second dataset onward, the time required for processing each dataset equals the maximum of τ_i. Hence, the total time for processing the kth datasets are given by:

$$T_{\text{pipe}} = T_{\text{circ}} + (k-1)\tau_{\text{max}}. \tag{14}$$

We call this the "steady-flow" pipelining scheme, as the data stream can flow through the entire pipeline without requiring modifications to the time gap between adjacent datasets or being put on hold at any point along the pipeline. Deviating from the steadyflow scheme, we often need to tempolary put

the dataset on hold in the pipeline. For this, we need to implement "baffer" where multiple datasets can be on hold.

6.2 Looped Qubit Pipeline

(a)

T_{circ}

τ_{gap}

Linear qubit pipeline

(b)

T_{loop}

τ_{gap}

Initialization or readout devices

Gate devices

Shuttling track

Qubits

Looped qubit pipeline

Fig. 14

Similar to the classical pipeline, we can define the linear qubit pipeline, as shown in Fig. 14(a). The qubits are processed at each step, which consists of shuttling, gate operations, initialization, or measurement. We can apply the steady-flow scheme to this pipeline, so the time required to process k qubits is given by Eq. 14. Note that, in practice, the rate-limiting step is often not the shuttling step, as the shuttling operation is relatively faster compared to other operations. Without loss of generality, we assume that the shuttling operation can hold the qubits without pushing them forward, allowing it to act as a buffering region for the pipelining when needed.

Beyond the linear qubit pipeline, we can define the looped qubit pipeline, as shown in Fig. 14. Unlike the linear qubit pipeline, where the circuits that can be performed on the qubits are restricted by the number of devices on the pipeline, the looped pipeline allows the qubits to circulate around the loop and reuse the gate devices. This enables the effective execution of circuits with any depth on the qubits. Focusing on the first qubit in the qubit stream, the time for wrapping around the loop is called the "cycling period," denoted as T_{loop}. Let us assume that all the additional qubits in the pipeline follow behind with a constant time gap τ_{gap} between consecutive qubits. To avoid a qubit collision, where the first qubit collides with the last qubit after wrapping around the loop, we need to ensure that the cycling period T_{loop} is larger than the time gap between the first qubit and the last qubit, which is $(k-1)\tau_{\text{gap}}$, where k is the number of qubits:

$$T_{\text{loop}} \geq (k-1)\tau_{\text{gap}}. \tag{15}$$

20

The cycling period T_{loop} may vary from one round to another because, for example, we might apply four gates in one round but only three gates in the next round, or certain rounds may not involve measurement or initialization. Let us denote the "minimum cycling period" throughout the entire pipeline process as $T_{\text{loop}}^{\text{min}}$. To avoid collisions, the following condition must hold:

$$T_{\text{loop}}^{\text{min}} \geq (k - 1)\tau_{\text{gap}}, \tag{16}$$

$$k \leq T_{\text{loop}}^{\text{min}}/\tau_{\text{gap}} + 1 = K_{\text{loop}} \tag{17}$$

where K_{loop} represents the maximum number of qubits that can fit in the pipeline. In the steday-flow scheme, we have $\tau_{\text{gap}} = \tau_{\text{max}}$. thus, the maximum number of qubits that can fit in the loop in this case is:

$$K_{\text{loop}} = T_{\text{loop}}^{\text{min}}/\tau_{\text{max}} + 1. \tag{18}$$

We can increase the number of qubits in the pipeline by reducing τ_{max}. When the rate-limiting step is measurement or initialization, we can reduce their effective processing time by adding more initialization or measurement devices, as shown in Fig. 15, and operating them in parallel. If there are m times more initialization or measurement devices in the pipeline, the rate-limiting time τ_{max}, which represents the effective measurement or initialization time in parallel, will be reduced by a factor of m, provided we are operating on more than m qubits: $\tau_{\text{max}} = \tau_{\text{init/meas}}/m$. This makes it possible to reduce τ_{max} to the extent that measurement or initialization is no longer the rate-limiting step.
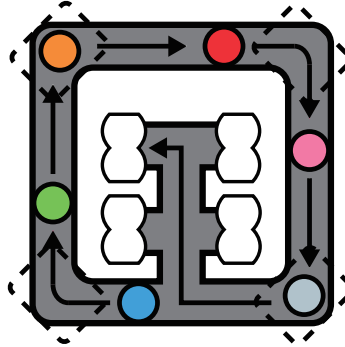

Fig. 15

7 Pseudo-three dimensional Surface Code

In this section, we describe the pseudo three-dimensional Surface Code on the looped pipeline architecture introduced in Section 6. First, we describe how computation is performed on multiple 2D Surface Codes in a processor. Then, we extend this concept into a pseudo three-dimensional structure with a periodic cycle in the direction of the third dimension.

7.1 Quantum Processor

In fault-tolerant quantum computation, the Surface Code, introduced in Section 4, is the most promising error correction code for the calculations required in many quantum algorithms. On the other hand, quantum low-density parity-check codes (qLDPC) are often considered more suitable for quantum memory due to their high encoding rate. However, while a single Surface Code can encode only one logical qubit, it offers many advantages, such as a simple approach for universal logical operations using lattice surgery combined with magic state distillation.

When designing the processor for computation, we simplify a single Surface Code into a "patch," which features dashed and solid lines. Simply put, a patch represents a logical qubit. In Fig. 16(a), three patches are allocated on the processor, and the corresponding Surface Codes are shown in Fig. 16(b), which are numbered. The rest of the qubits in Fig. 16(b) are unused data qubits for lattice surgery, as introduced in Section 5.
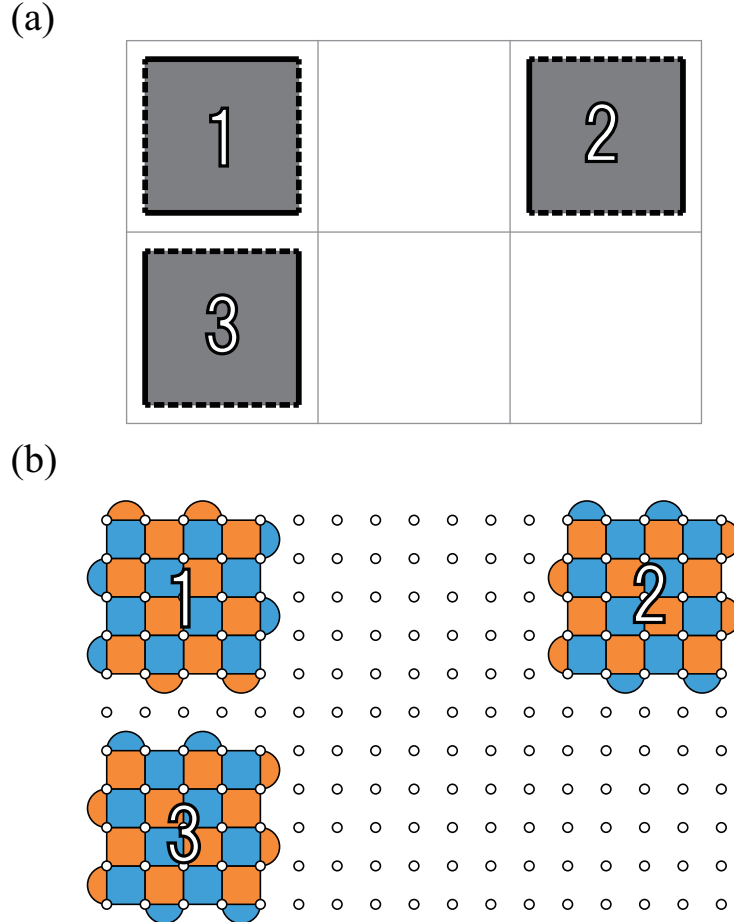
(a)

(b)

Fig. 16

Using lattice surgery, we can perform logical operations between two patches, three patches, or more. Additionally, we can perform commutative surgery operations in parallel when there exists a route from the control qubit to the target qubit by using unused data qubits in the processor. In this scheme, the efficiency of computation depends on how many parallel operations we can execute, thus requiring careful decision-making regarding the routing of operations. For instance, a certain logical 2-qubit operation between patch 2 and patch 3 is shown in Fig. 17. In this case, we cannot perform a logical operation between patch 1 and patch 2 in parallel.

(a)

(b)
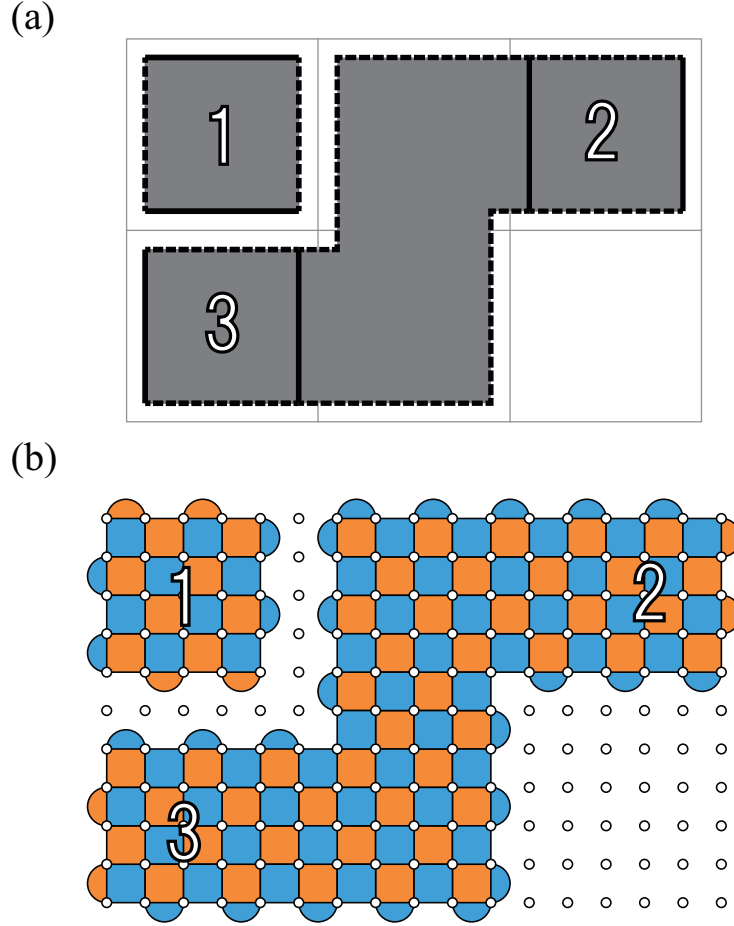
Fig. 17

In Fig. 18, some logical operations performed in parallel are illustrated with blue routes, while the orange route is prohibited since it intersects with an existing blue route at their intersection.
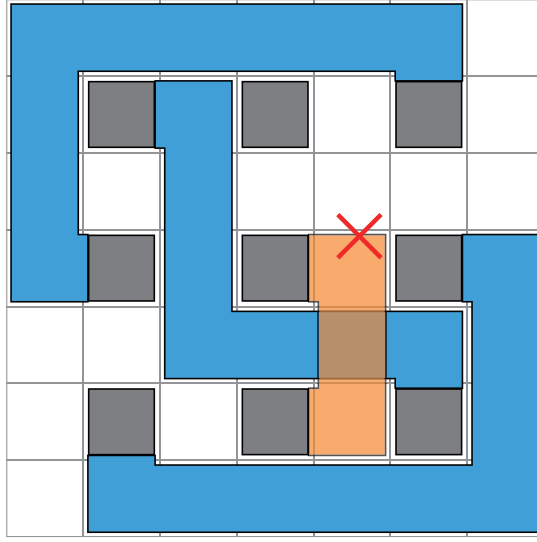
Fig. 18

7.2 Pseudo Three-dimensional Surface Code

First, we introduce the Surface Code implemented in the looped pipeline as described in Section 6. In the looped pipeline architecture, we consider a single looped pipeline as a qubit in the Surface Code, as shown in Fig. 19. The red dashed lines represent a two-qubit gate for the two physical qubits in the looped pipeline. In the following description, the devices for measurement and initialization are not shown for simplicity.
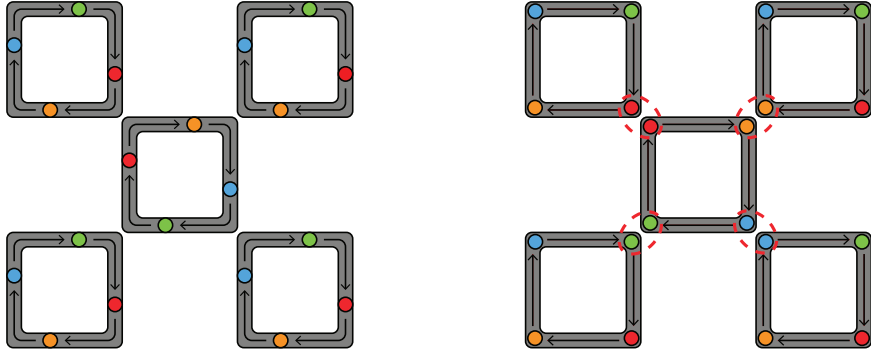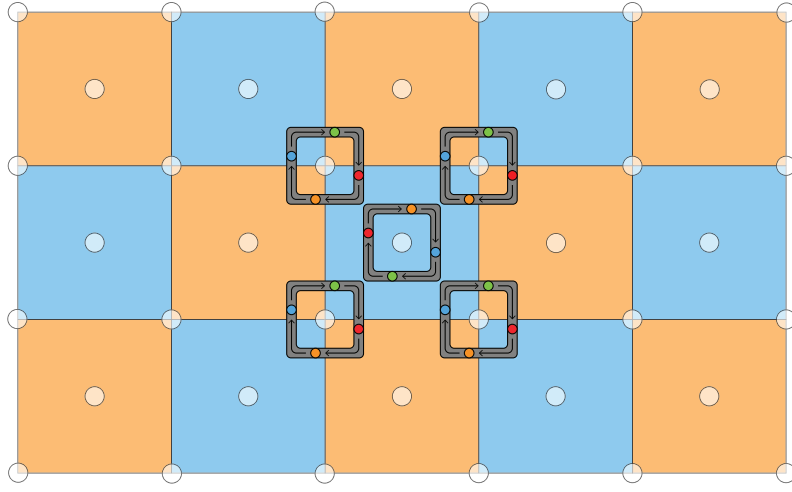

Fig. 19

Considering the looped pipeline in the middle of the five pipelines as the ancilla qubit for syndrome measurement, and the looped pipelines surrounding the ancilla qubit as the data qubits, we can see that the allocation of looped pipelines corresponds to the Surface Code, as shown in Fig. 20. There exist synchronized qubits with the same color in the looped pipeline.

Fig. 20

In addition to the interloop interactions shown in the right half of Fig. 19, we can perform intraloop interactions as shown in Fig. 21.
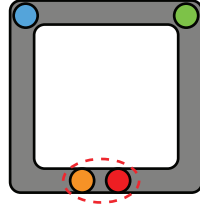
Fig. 21

Now, we have four qubits in a single pipeline; thus, we have four stacks of Surface Codes, as shown in Fig. **??**. In the following description, we refer to each stacked Surface Code as a "layer." Furthermore, when representing the pseudo-3D Surface Code as shown in Fig. 22, we refer to each layer as a "floor." For example, we designate the red layer as floor 0 and the blue layer as floor 2. On the one hand, by introducing interloop interactions, as shown in Fig. 20, we enable the construction of the Surface Code. On the other hand, by introducing intraloop interactions, we can perform logical two-qubit operations between qubits, each present in adjacent layers. Lastly, it is worth noting that in Fig. 22, floor 0 and floor 3 are adjacent since the red qubits and orange qubits are adjacent in the looped pipelines.


looped pipeline   pseudo-3D surface code
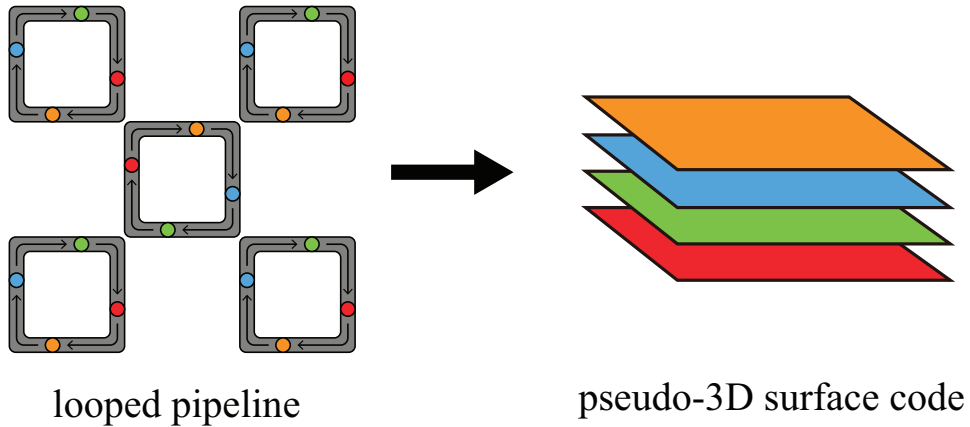
Fig. 22

7.3 Lattice Surgery Routing

In this subsection, we will show the advantage with using the psuedo-3D surface code for the lattice surgery routing. The numerical results are shown in Section.9.

In Fig. 23, we are showing the distance advantage of the pseudo-3D Surface Code. When the blue operation in Fig. 23 is being processed, the orange operation needs to make a detour in the 2D Surface Code. However, in the pseudo-3D Surface Code, the orange operation can bypass the blue operation by utilizing the upper layer. Thus, the total cost for routing the blue and orange operations is less in the pseudo-3D Surface Code than in 2D.
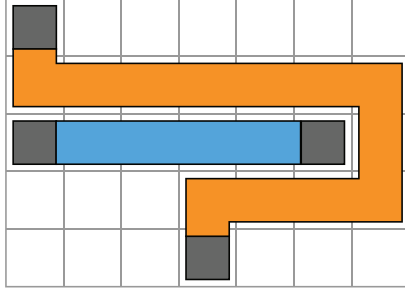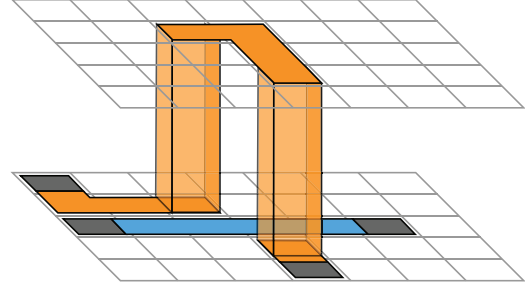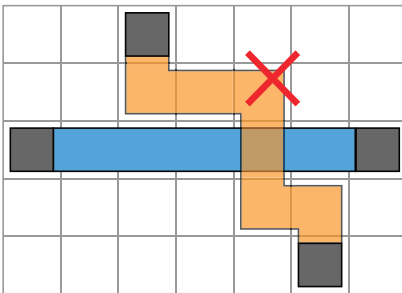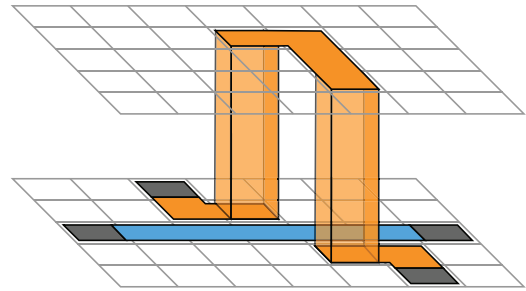

2D surface code   pseudo-3D surface code

Fig. 23

In Fig. 24, we are showing the parallelization advantage of the pseudo-3D Surface Code. When the blue operation in Fig. 24 is being processed, the orange operation cannot be executed in the 2D Surface Code because there is no available route for the orange operation. However, in the pseudo-3D Surface Code, the orange operation can bypass the blue operation by utilizing the upper layer. Thus, the total time for executing the blue and orange operations is reduced in the pseudo-3D Surface Code compared to the 2D Surface Code.

2D surface code   pseudo-3D surface code

Fig. 24

8 Placement Optimization

In this section, we describe the placement optimization for the patches in the quantum processor. Numerical results are shown in Section 9.

8.1 Mapping the Circuit to the Graph

We consider placement optimization based on the number of surgery operations in the quantum circuit. Firstly, we count the number of surgery operations, e.g., multi-body Pauli measurements or CNOT gates, in the circuit. Then, we construct a graph $G(V, E)$ as follows:

Definition 2. *Let V represent the nodes and E represent the edges in the graph $G(V, E)$. Each patch in the quantum processor and each two-qubit operation are mapped to V and E, respectively. Additionally, each edge has a weight w, which corresponds to the number of operations associated with the edge.*

Particularly, we denote the patch used to implement T gates as `MAGIC_NODE` in the graph. We assume that when performing a T gate on a qubit, we execute a two-qubit operation between the qubit and `MAGIC_NODE` instead of performing the T gate directly on the qubit. Now, we present an example of mapping a small circuit. In Table 3, the names of the qubits are shown in the first column, and the instruction numbers, control qubits, target qubits, and weights are shown in the second column. And the result of the mapping Table 3 is shown in Fig.25. The coordinates of the qubits are on the grid. `MAGIC_NODE` is fixed at the certain place where magic state factories exists.
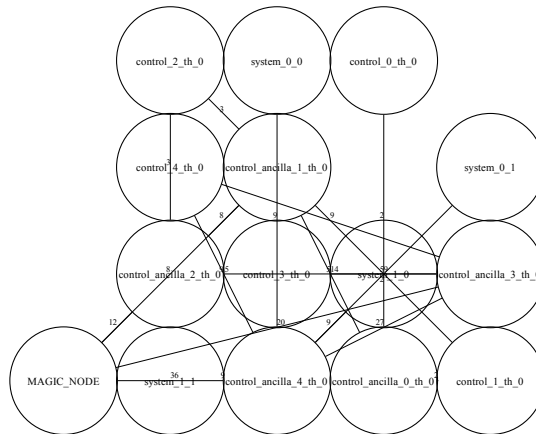

Fig. 25

Table 3

qubits (nodes)	instruction_number control target weight (edges)
MAGIC_NODE	1 system_1_1 control_ancilla_4_th_0 9
control_0_th_0	2 control_ancilla_0_th_0 control_1_th_0 2
control_1_th_0	3 control_ancilla_2_th_0 control_3_th_0 5
control_2_th_0	4 system_1_0 control_ancilla_4_th_0 9
control_3_th_0	5 control_ancilla_3_th_0 control_4_th_0 9
control_4_th_0	6 control_ancilla_4_th_0 control_4_th_0 9
control_ancilla_0_th_0	7 control_ancilla_1_th_0 control_1_th_0 2
control_ancilla_1_th_0	8 MAGIC_NODE control_ancilla_2_th_0 12
control_ancilla_2_th_0	9 control_ancilla_3_th_0 control_ancilla_2_th_0 14
control_ancilla_3_th_0	10 system_0_1 control_ancilla_4_th_0 9
control_ancilla_4_th_0	11 control_ancilla_2_th_0 control_2_th_0 3
system_0_0	12 MAGIC_NODE control_ancilla_3_th_0 20
system_0_1	13 system_0_0 control_ancilla_4_th_0 9
system_1_0	14 control_ancilla_2_th_0 control_ancilla_1_th_0 8
system_1_1	15 control_ancilla_1_th_0 control_2_th_0 3
	16 control_ancilla_4_th_0 control_ancilla_3_th_0 27
	17 control_ancilla_0_th_0 control_0_th_0 2
	18 control_ancilla_3_th_0 control_3_th_0 5
	19 control_ancilla_1_th_0 control_ancilla_0_th_0 5
	20 MAGIC_NODE control_ancilla_4_th_0 36
	21 MAGIC_NODE control_ancilla_1_th_0 8

8.2 Potential Energy Model

In this subsection, we introduce the potential model applied to the graph and allocate the qubits on the grid. Assume that there exists an edge between qubit 1 and qubit 2, with coordinates denoted as (x_1, y_1) and (x_2, y_2) on the grid. The distance between qubit 1 and qubit 2 is defined as $d = |x_1 - x_2| + |y_1 - y_2|$, representing the Manhattan distance. Thus, the potential energy $p(w, d)$ is defined using the weight of the edge and the distance.

Definition 3. *Assume that w is the weight of an edge in the graph $G(V, E)$ and d is the Manhattan distance between the two qubits. Thus, the potential energy $p(w, d)$ between the two qubits is defined as:*

$$p(w, d) = wd^2. \tag{19}$$

From Definition 3, we can calculate the total potential energy P for all edges in the graph. Thus, the following equation holds:

$$P = \sum_{i \in \text{edges}} p_i(w, d). \tag{20}$$

The procedure for updating the allocation of qubits is as follows. First, select a qubit based on the descending order of the total weight of edges associated with each qubit. Second, randomly choose another qubit within a Manhattan distance of l from the first qubit to consider for exchange. If the total potential energy decreases as a result of the exchange, perform the swap of these qubits. Otherwise, retain their current positions without exchanging them and return to the first step. This procedure continues until the potential energy becomes sufficiently small. In Fig. 26, the results obtained by reordering the qubits as shown in Fig. 25 and using the potential energy defined in Eq. 20 are presented.
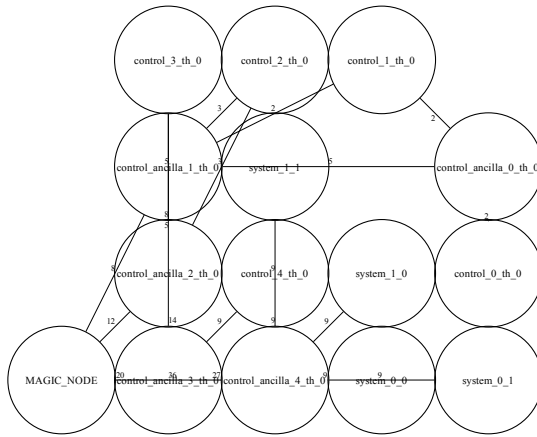
Fig. 26

This scheme can be expanded to three dimensions by redefining the distance metric d as: $d = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$, where the coordinates of qubits are denoted by (x, y, z).

9 Results

Firstly, we perform numerical simulations of the pseudo-3D Surface Code, introduced in Section **??**, using a circuit that executes a 2D Heisenberg model and compare it to the 2D Surface Code. The results are presented in Fig. 27, and a more complex system is shown in Fig. 28. In these figures, the horizontal and vertical axes represent the instruction number and distance, respectively, where distance refers to the number of patches required for routing operations. Additionally, the 20 magic patches that produce T gates are aligned along $x = -1$, arranged from $y = 0$ to $y = 19$. The Breadth-First Search algorithm is used for routing in the following cases.
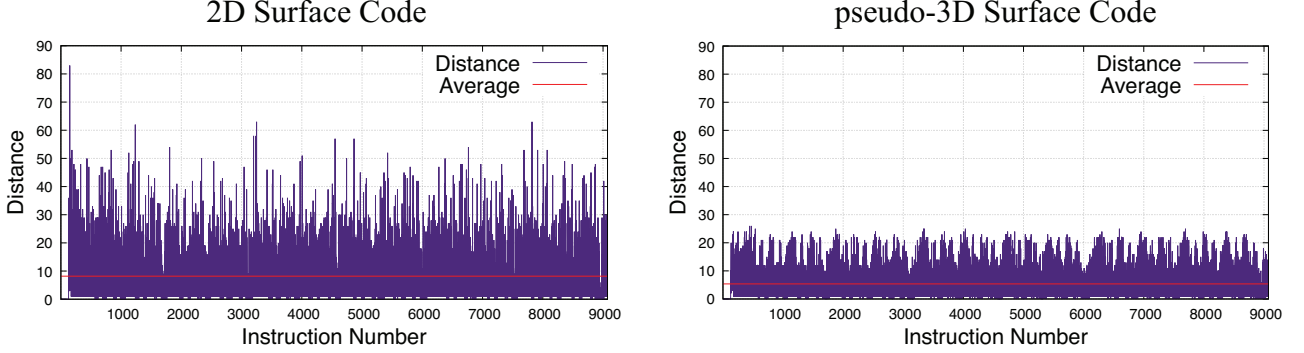

Fig. 27

In Fig. 27, the graph illustrates that in the 2D Surface Code, a substantial number of operations exceeded a distance of 30. In contrast, in the 3D Surface Code, the longest operation distance remained below 30. Furthermore, the average distance per operation was significantly reduced from 8.18 in the 2D configuration to 5.36 in the pseudo-3D configuration, resulting in an approximately 66% improvement in overall circuit distance. However, the total time required to execute the entire circuit did not change despite expanding the routing dimension to pseudo-3D.
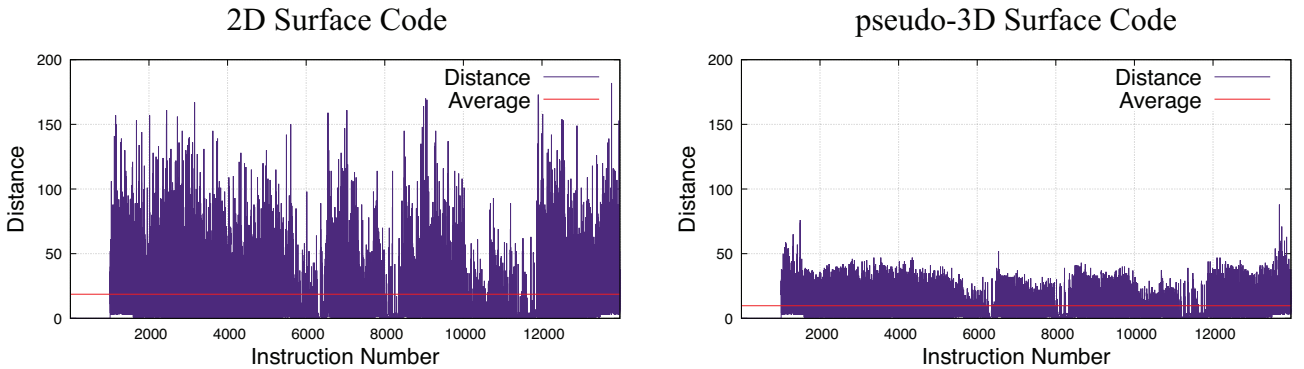

Fig. 28

In the more complex circuit, where the Heisenberg model system is the same as in the former case but is more parallelized and thus requires more ancilla qubits, similar improvements are observed, as shown in Fig. 28. This further demonstrates the effectiveness of the pseudo-3D Surface Code in reducing operation distances. The average distance per operation was significantly reduced from 18.64 in the 2D configuration to 9.74 in the pseudo-3D configuration, resulting in an approximately 52% improvement in overall circuit distance. However, the total time required to execute the entire circuit

did not change in the more complex system. Therefore, the pseudo-3D Surface Code does not improve the parallelization of the circuit.

Secondly, we perform numerical simulations of the placement optimization method introduced in Section 8. The results are presented in Fig. 29 for the 2D case and Fig. 30 for the pseudo-3D case. Additionally, the graphs optimized by the potential energy model are shown in Fig. 31 and Fig. 32. Each figure in Figs. 29–32 displays the results of a smaller circuit in the top row and a larger circuit in the bottom row. The system for the Heisenberg model is the same as in the previous case; thus, the results on the left-hand side of Figs. 29–32 have already been discussed in the preceding paragraph. The 20 magic patches that produce T gates are aligned along $x = -1$, arranged from $y = 0$ to $y = 19$, while logical patches are allocated in the range of $x \geq 0$ and $y \geq 0$. The parameter l, introduced in Section 8, was tuned to a value of 3 in all the following cases, and the Breadth-First Search (BFS) algorithm is used for routing in the following cases.

not optimized in 2D	optimized in 2D
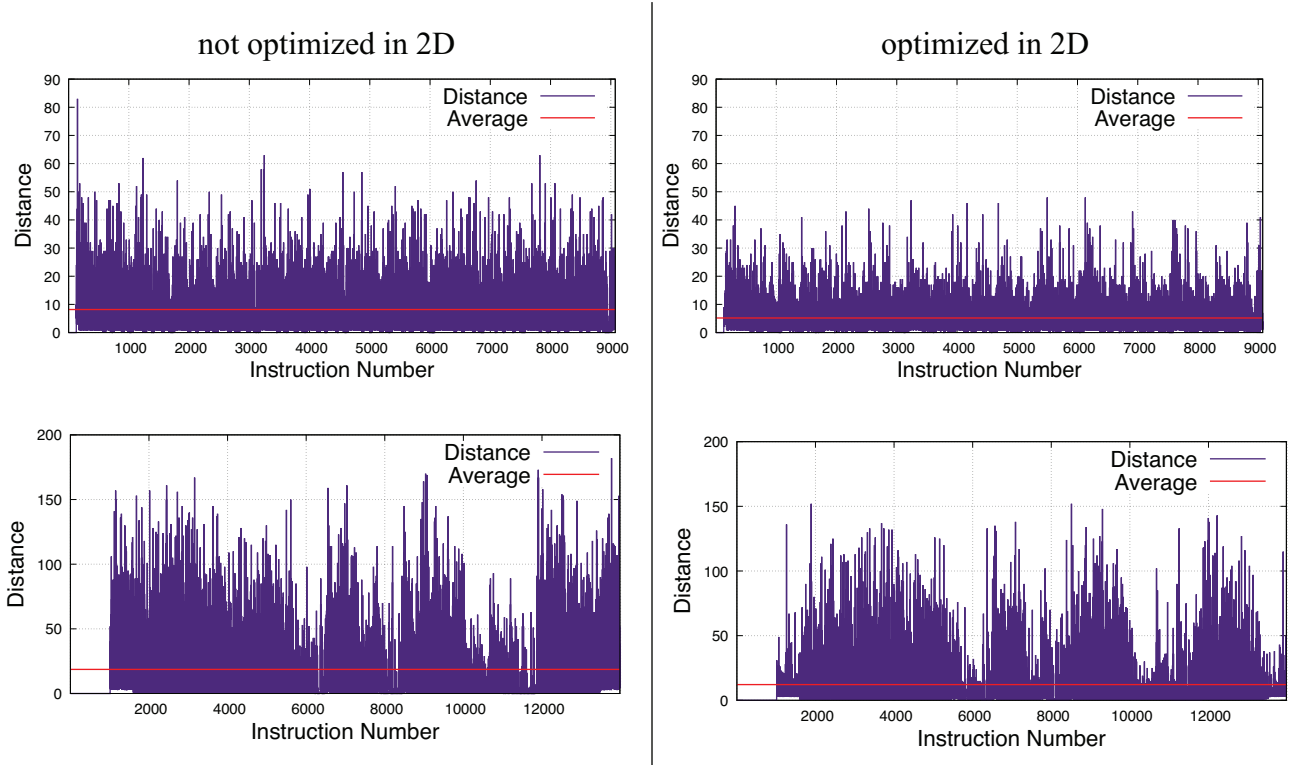
Fig. 29

In Fig. 29, we obtained results showing that placement optimization decreases routing distances. Specifically, the average distance in the 2D optimized configuration was reduced to 5.18 for the smaller circuit and to 12.15 for the larger circuit, resulting in a 63% improvement in the smaller circuit and a 65% improvement in the larger circuit.
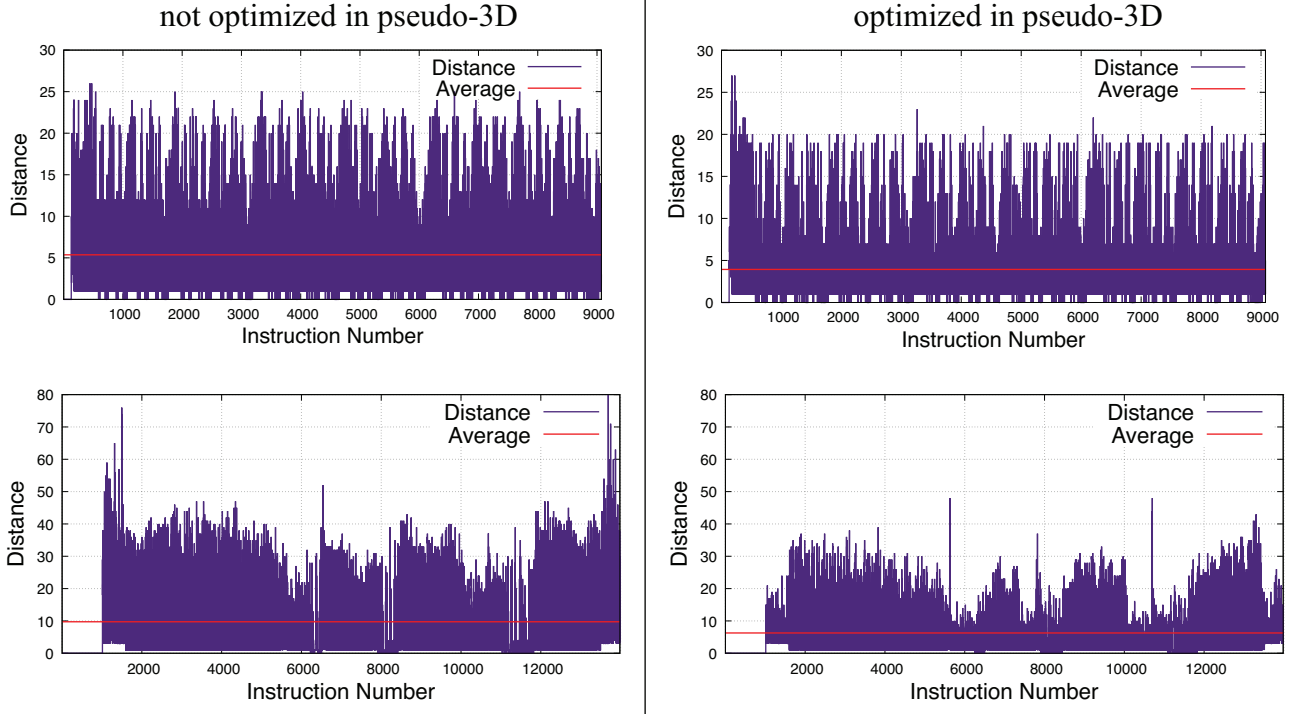
Fig. 30

Similarly, in Fig. 30, we observed that placement optimization effectively decreases routing distances in the pseudo-3D configuration. Specifically, the average distance in the pseudo-3D optimized configuration was reduced to 3.93 for the smaller circuit and to 6.28 for the larger circuit, resulting in a 73% improvement in the smaller circuit and a 64% improvement in the larger circuit.

Throughout the all results, the distances needed to routing is more effectively reduced in the larger cicuit than that in smaller circuit. But there exists a trade-off that the BFS algorithm is more complicated in pseudo-3D, thus the routing time cannot be ignored in a larger circuit in the pseudo-3D. Moreover, placement optimization time in a graph also cannot be ignored in a larger circuit in the pseudo-3D.
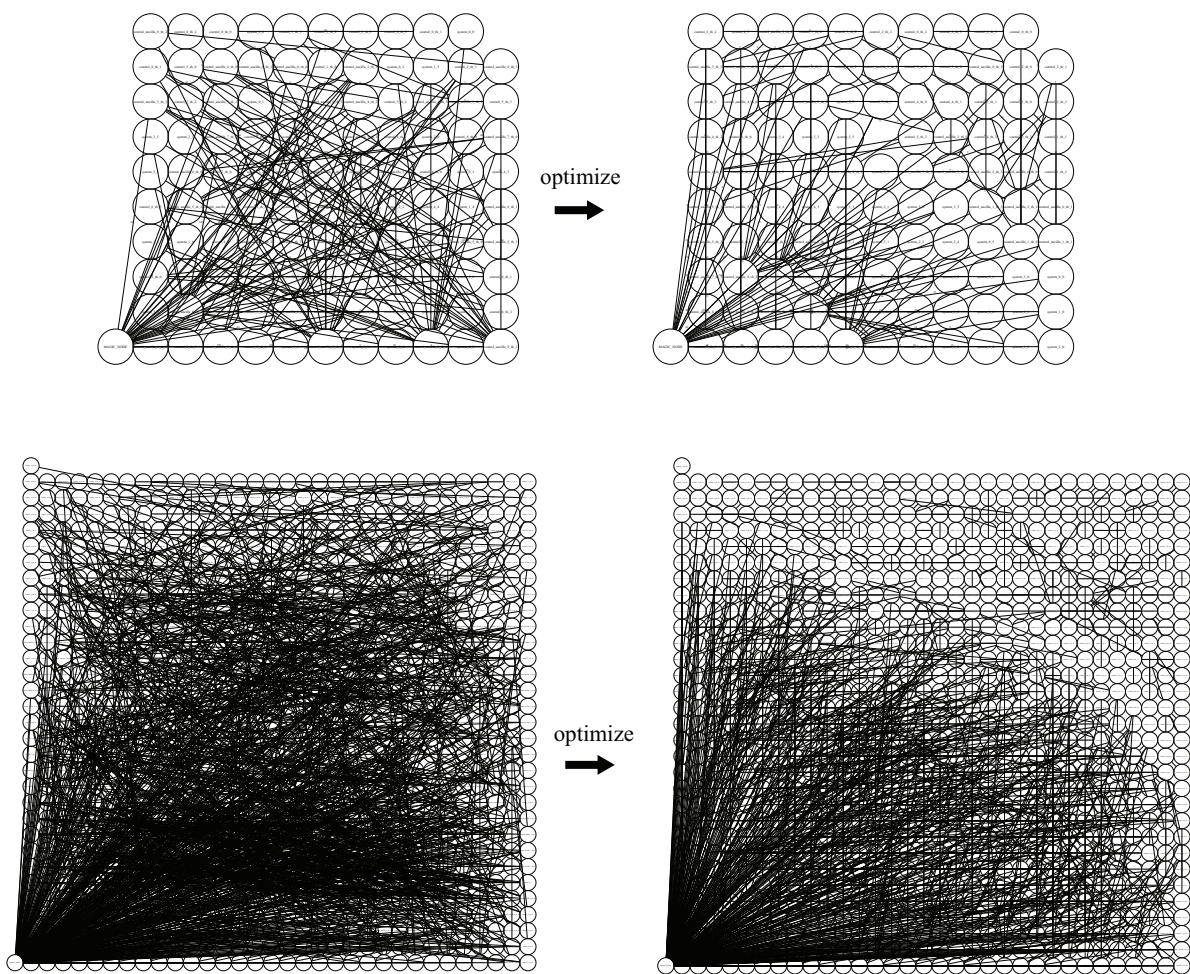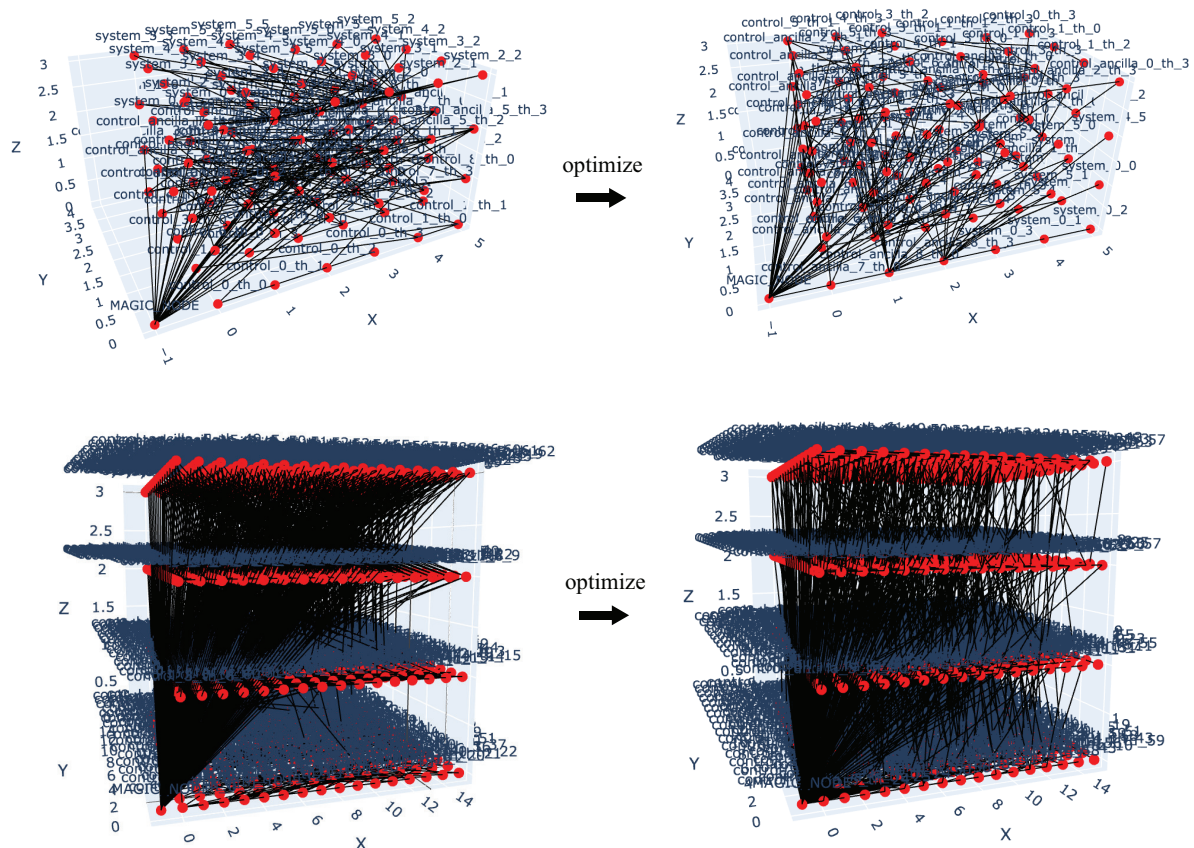
Fig. 31

Fig. 32

10 Conclusion and Future Works

In this paper, we introduced the pseudo-three-dimensional surface code utilizing the looped pipeline architecture in semiconductor quantum computers. Additionally, we presented a placement optimization method based on the potential energy between logical qubits in the quantum processor. Our numerical results demonstrate an approximate 60% improvement in routing distance, indicating that the looped pipeline can enhance computational efficiency despite the physical qubits being confined to two dimensions on the processor. Moreover, similar improvements are anticipated in ion-trapped quantum computers. By performing classical processing, such as placement optimization, we achieved approximately 63% improvement in 2D configurations and 69% improvement in pseudo-3D configurations. Since our placement optimization method imposes no restrictions regarding the type of quantum computer, this scheme can be applied to any quantum processor, and similar enhancements can be expected.

Throughout this paper, we have not been able to improve the time required to execute the circuit using the pseudo-three-dimensional surface code. Therefore, optimizing execution time remains an area for future work. Additionally, we introduced a new method for placement optimization based on potential energy. However, there is room for further improvement, such as refining the parameter l, adjusting the exponent of d in Eq. 19, and exploring other related factors.

Acknowledgments

REFERENCES

[1] Dolev Bluvstein, Simon J. Evered, Alexandra A. Geim, Sophie H. Li, Hengyun Zhou, Tom Manovitz, Sepehr Ebadi, Madelyn Cain, Marcin Kalinowski, Dominik Hangleiter, J. Pablo Bonilla Ataides, Nishad Maskara, Iris Cong, Xun Gao, Pedro Sales Rodriguez, Thomas Karolyshyn, Giulia Semeghini, Michael J. Gullans, Markus Greiner, Vladan Vuletić, and Mikhail D. Lukin. "Logical Quantum Processor Based on Reconfigurable Atom Arrays", Nature **626**, 58–62. DOI: `10.1038/s41586-023-06927-3`. URL: `https://doi.org/10.1038/s41586-023-06927-3`.

[2] Google Quantum AI and Collaborators. "Quantum Error Correction Below the Surface Code Threshold", arXiv preprint. eprint: `2408.13687`. URL: `https://arxiv.org/abs/2408.13687`.

[3] Dominic Horsman, Austin G. Fowler, Simon Devitt, and Rodney Van Meter. "Surface code quantum computing by lattice surgery", New J. Phys. **14**, 123011. DOI: `10.1088/1367-2630/14/12/123011`. URL: `https://doi.org/10.1088/1367-2630/14/12/123011`.

[4] Daniel Litinski. "Magic State Distillation: Not as Costly as You Think", Quantum **3**, 128. URL: `https://quantum-journal.org/papers/q-2019-12-02-205/`.

[5] Craig Gidney, Noah Shutty, and Cody Jones. "Magic state cultivation: growing T states as cheap as CNOT gates", arXiv preprint. eprint: 2409.17595. URL: https://arxiv.org/abs/2409.17595.

[6] Zhenyu Cai, Adam Siegel, and Simon Benjamin. "Looped Pipelines Enabling Effective 3D Qubit Lattices in a Strictly 2D Device", PRX Quantum **4**, 020345. DOI: 10.1103/PRXQuantum.4.020345. URL: https://doi.org/10.1103/PRXQuantum.4.020345.

[7] Simon J. Devitt, William J. Munro, and Kae Nemoto. "Quantum Error Correction for Beginners", arXiv preprint. eprint: 0905.2794. URL: https://arxiv.org/abs/0905.2794.

[8] Daniel Gottesman. "The Heisenberg Representation of Quantum Computers", arXiv preprint. eprint: quant-ph/9807006. URL: https://arxiv.org/abs/quant-ph/9807006.

[9] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information.* 10th Anniversary Edition. Cambridge, UK: Cambridge University Press, 2010. ISBN: 978-1-107-00217-3. URL: https://www.cambridge.org/9781107002173.

[10] A. M. Steane. "Simple Quantum Error Correcting Codes", arXiv preprint. eprint: quant-ph/9605021. URL: https://arxiv.org/abs/quant-ph/9605021.

[11] A. Yu. Kitaev. "Fault-Tolerant Quantum Computation by Anyons", arXiv preprint. eprint: quant-ph/9707021. URL: https://arxiv.org/abs/quant-ph/9707021.